# ATLAS distributed analysis

D.L. Adams, W. Deng, N. Chetan, C. Kannan, V. Sambamurthy, K. Harrison, C.L. Tan, A. Soroko, D. Liko, F. Orellana, et al.

# ATLAS DISTRIBUTED ANALYSIS

D. L. Adams, W. Deng, BNL, Upton, NY, USA
N. Chetan, C. Kannan, V. Sambamurthy, SUNY, Stony Brook, NY, USA
K. Harrison, University of Cambridge, Cambridge, UK
C. L. Tan, University of Birmingham, Birmingham, UK
A. Soroko, Oxford University, Oxford, UK
D. Liko, F. Orellana, M. Branco, CERN, Geneva, Switzerland
C. Haeberli, University of Bern, Bern, Switzerland
S. Albrand, J. Fulachier, LPSC, Grenoble, France
J. Lozano, F. Fassi, IFIC, Valencia, Spain
G. Rybkine, Royal Holloway College, University of London, Egham, UK

*Abstract*

The distributed analysis system for the ATLAS experiment is described. Although the system is under construction and the design is still evolving, the major components have been identified. A generic analysis service interface provides the connection between a choice of user environments and a selection of engines capable of carrying out the distributed processing. Other services provide access to catalogs, files and software.

## INTRODUCTION

ATLAS [1] is a large high-energy physics collaboration constructing a detector that will acquire data from collisions taking place in the Large Hadron Collider (LHC) at CERN. ATLAS will read a few petabytes per year from the detector into an offline production system that will produce a similar amount of data.

### *Goals*

ADA [2], the ATLAS Distributed Analysis system, will enable ATLAS physicists to analyze this data and to carry out production of moderately large samples. Another important goal is to provide provenance tracking, i.e. to provide a reliable and comprehensible record of the chain of processing behind any piece of data. Finally, the system must perform well and be easy to use and access from the expected analysis environments (python [3] and ROOT [4]) and flexible enough to adapt to other environments that might later be of interest (e.g. JAS [5]).

### *Status*

ADA is under active development and this note describes a snapshot of the current status, a summary of our plans for this year and some thoughts on later evolution. Much of the ADA model and underlying software is inherited from DIAL [6]. A user interface is being developed by GANGA [7] and metadata services by AMI [8].

### *Applicability*

Although ADA is being developed in the ATLAS context, the underlying software packages such as DIAL, GANGA and AMI have little dependence on the ATLAS software. Care is taken to keep the entire system generic to allow use in other contexts that provide the required data and application wrappers.

## JOB DEFINITION LANGUAGE

To facilitate provenance tracking and provide a friendly interface, ADA makes use of a high-level job definition language to describe the data and its processing. This AJDL (Abstract Job Definition Language) [9] provides generic definitions of the fundamental processing components: datasets, their transformations and the jobs that carry these out.

### *Properties*

In order to determine the properties of these components, we examine how they are used both by our analysts and by a processing system. We distinguish between the inherent properties of a component and the metadata associated with it. The former are stored in XML strings that describe these objects while the latter are held externally, e.g. in relational tables.

A fundamental property of all components is their identity. It must be unique within the relevant scope to ensure there is no ambiguity in the provenance chains and other references expressed in terms of these identities.

The AJDL components and their properties are summarized in table 1. The following sections provide details for the individual components.

Table 1. Summary of AJDL types and their properties.

| | |
|---|---|
| Dataset | ID, location, content<br>Sub-dataset ID's |
| Transformation | ID, application ID, task ID |
| Application | ID, build_task script, run script |
| Task | ID, embedded named files |
| Job | ID, transformation ID, input dataset ID, result dataset ID, sub-job ID's, job history parameters |
| Job preferences | ID<br>List of name-value pairs |

### *Mutability*

We say that an object is immutable if its inherent properties cannot be altered. We identify three interesting states of mutability: immutable, fully mutable and extensible. In the latter case, new properties may be added but existing properties may not be changed. Most objects will be immutable to maintain the integrity of provenance chains. Some objects are mutable or extensible temporarily during construction.

### *Datasets*

Analysis proceeds as a series of steps where some action is taken on a collection of data to produce another collection. We call such a collection a dataset [10]. Note that the data itself is not one of these properties—when we speak of a dataset, we mean a description or specification of that data and not the actual data.

One natural property of a dataset is its location, i.e. the location of the actual data. This location may be expressed in many ways; one of the most common being a list of logical files. We allow for the possibility that a dataset does not have a location and call such a virtual dataset. A virtual dataset may not yet have a concrete representation or it may serve as an index into a dataset replica catalog.

Another important dataset property is content, meaning a description of the type of data it holds. For particle and nuclear physics applications, a particularly important class of dataset is the event dataset which holds the data associated with a series of events. A complete description of the content for such a dataset includes the list of event identifiers and a description of the type of data for each event.

Datasets are hierarchical, i.e. one of their properties is a list of contained datasets. This structure provides natural boundaries for dataset splitting. ATLAS plans to construct dataset hierarchies with the lowest-level datasets corresponding to single files.

### *Dataset operations*

Operations to construct datasets include creation, splitting, merging, and transformation. A dataset may be created from any collection of data and one common activity is to construct a dataset from all the data in a single file. Splitting distributes the data from one dataset over multiple datasets while merging does the converse. Transformation is defined to be an operation that takes one dataset as input and produces one as output.

### *Transformations*

The only operation provided by the distributed processing system of ADA is the transformation. There is implicit splitting of the input dataset and, after processing, merging to create the output dataset. The transformation is split into two sub-components: the application that provides a wrapper around the domain-specific (i.e. ATLAS) software and the task that carries data used to configure the application.

At present, the task carries a collection of embedded named text files. We anticipate adding named parameters and providing support for logical files.

The present definition of the application includes two entry points (scripts): build_task which is used to prepare the task and run which carries out the actual transformation.

The essential provenance of a dataset is specified by the input dataset and transformation used to produce that dataset. For physical datasets, these (and other job conditions) are recorded as part of the job history. If a user submits a virtual dataset, then the processing system will typically construct a virtual dataset from the output dataset. The mapping from input virtual dataset and transformation to output virtual dataset is recorded a virtual data catalog. This catalog is similar to the one that appears in Chimera [11].

It may be useful to add specification of input and output dataset content to the transformation to allow users and the processing system to discover incompatibilities between an input dataset and a transformation.

### *Jobs*

A job is a particular instance of a transformation acting on a dataset. The properties of a job include its definition (transformation and input dataset), its state (submitted, running, done, failed, etc.) and its history (start and stop times, computer where it was run, CPU, Memory and I/O consumption, etc.). A completed job or one with a partial result provides a link to its output dataset.

Jobs are hierarchical with a sub-job structure corresponding to the substructure of the input dataset. The structure may be even more complicated if sub-jobs are resubmitted after failure or additional jobs are introduced to carry out splitting or merging.

### *Job Preferences*

AJDL also provides a job preferences component that allows users some control over job processing. Examples include hints or requirements for data placement, splitting and merging and fault handling. Or a user might supply constraints for response times or a maximum budget for the processing. The job preferences are not part of the essential provenance and are stored as part of the job.

### *XML representation*

ADA makes use of the present DIAL implementation of AJDL. DIAL defines each component as a C++ class or class hierarchy. Each class provides means to write out to and read back from an XML description. We plan to formalize the definition of AJDL in terms of XML schema in collaboration with GANGA and others.

### *Services*

We have identified a number of services that act upon the AJDL components and are standardizing their interfaces in WSDL.

## ANALYSIS SERVICE

DIAL defines a C++ interface called scheduler for defining and running jobs. This interface provides methods for installing applications and tasks and submitting and monitoring jobs. DIAL also defines a web service interface called analysis service with similar functionality, and provides wrappers using this interface to enable requests to be relayed from a client scheduler on one machine to a scheduler running inside a web service on a remote node.

ADA has adopted the analysis service as the interface by which clients submit requests for processing. Figure 1 shows the initial plan for the ADA architecture. There are three major options for the analysis service, based on DIAL, ATLAS production [12] and ARDA [13].

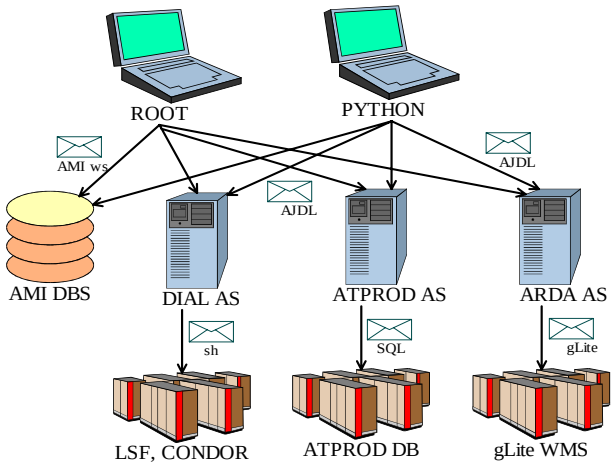


Figure 1. High level view of the ADA architecture showing the two types of clients, the catalog services and the three major analysis services

### *DIAL*

DIAL provides schedulers for local processing (fork) and for batch submission to LSF [14] or Condor [15]. It is easy to add new job classes to describe other batch or workload management systems. An important goal of the DIAL project is to provide services with interactive response, i.e. those for which result begin to flow back within a few seconds after submission. Options for such processing include a specially tuned LSF queue and use of Condor COD (Computing on Demand).

### *ATLAS production system*

It is planned to add an analysis service which submits jobs to the ATLAS production system. A DIAL job subclass will be added to connect to the database that serves as the interface to that system.

### *ARDA*

ARDA is a project to deliver prototype distributed analysis systems for the four LHC experiments. These systems will be based on gLite [16], the EGEE middleware. For ATLAS, the prototype includes an analysis service submitting jobs to the gLite WMS workload management system). A simple implementation of this service exists for the initial gLite prototype. The first release of the gLite software is expected this year and the ARDA team will quickly follow with a corresponding analysis service. The first version of this service will also be based on DIAL, again by adding an appropriate job subclass.

The motivation for adding the ARDA service is the expectation that EGEE will deliver a robust and reliable workload management system capable of scaling to ATLAS requirements for a worldwide grid.

## CATALOG SERVICES

ADA recognizes the need for catalog services to provide persistence for AJDL objects, enable users to select from existing objects and to record replica and provenance information.

### *Types of services*

DIAL defines three C++ interfaces for cataloguing: repository, selection catalog and replica catalog. ADA makes use of all of three: repositories for all AJDL types and, at least for datasets, selection and replica catalogs. A repository stores XML object descriptions indexed by ID. A selection catalog associates named attributes (metadata) with object ID's. A replica catalog associates one logical ID with a collection of replica ID's.

### *Web services*

We would like to define web service interfaces for the three types of catalogs. AMI already provides a web service interface but it is a very generic catalog interface and we plan to supplement it with these more restrictive interfaces. The metadata catalog interface identified by the gLite is a candidate for the selection catalog.

## OTHER SERVICES

In addition to the catalog and analysis services visible to the user, processing requires access to services for data and software package management.

### *Data Management*

DIAL defines an interface called file catalog that provides an interface for storing and retrieving physical files identified with logical names. There are implementations based on a local filesystem, NFS, AFS and magda [17]. The latter is used for data from the first ATLAS data challenge. New production data is accessible though DQ (Don Quijote) [18], a client-service application developed within ATLAS. ADA is adopting DQ as its data management interface and a corresponding file catalog implementation will be added to DIAL.

### *Package management*

DIAL makes very few assumptions about the software used to carry out processing except that the software must be wrapped in the DIAL application interface. This wrapper (usually implemented as scripts) must locate

software when it runs on a worker node. Most of the current grid production systems preinstall software and then use environmental variables or an information service to advertise its location.

Both DIAL and ADA would like to move to a service-oriented model with a generic interface that could support multiple strategies for software management and enable installation on demand. This issue has been raised with ARDA and gLite. ADA is developing packagmgr [19] as an interim solution.

## USER INTERFACES

ADA provides its users with easy access to data and processing. It is integrated with ROOT and Python and provides graphical and command line clients.

### *ROOT*

DIAL is implemented in C++. The ROOT tool ACLiC has been used to parse the DIAL header files and construct a dictionary which makes all the DIAL classes available at the ROOT command line. Users can create and examine applications, tasks and datasets and use the scheduler interface to submit and monitor jobs.

### *Python*

ATLAS also supports python as an analysis framework. Analogous to the approach with ROOT, lcgdict from SEAL [20] has been used to parse the DIAL headers and construct dictionaries that map the DIAL classes to python classes. This has been done in the context of the GANGA project.

GANGA is also interested in delivering a lightweight client that removes the dependency on the DIAL C++ classes.

### *Graphical interface*

ADA will provide its users with a graphical interface to aid in examining data and specifying, submitting and monitoring jobs. This work is also being done by GANGA making use of the python binding to DIAL.

## CURRENT STATUS

ADA is operational but is in demo mode. Much of the persistency is missing and most of the transformations required for ATLAS have not yet been defined.

There are two analysis service instances running at BNL: one for long-running jobs and one that provides interactive response. ADA has clients to enable use of the AMI service but little of its data is resident there.

All of the reconstructed data from the first ATLAS data challenge are available as ADA datasets. A transformation uses PAW to create histograms from that data. The user task to configure this transformation includes a list of histogram definitions and a FORTRAN routine to do the event processing.

Figure 2 shows the transformations required to fully support the current ATLAS data model. Only a couple prototype transformations, DIGI and RECO, are in place.

| IN \ OUT | EVTIDS | EVGEN | HITS | DIGITS | RAW | ESD | AOD | TAG | NTUP | HISTO |
|---|---|---|---|---|---|---|---|---|---|---|
| | IDBLD | | | | DAQ | | | | | |
| EVTIDS | | GEN | | | | | | | | |
| EVGEN | | | G4SIM | | | | | | G4SIM | G4SIM |
| HITS | | | | DIGI | | | | | DIGI | DIGI |
| DIGITS | | | | | PACK | RECO | | | RECO | RECO |
| RAW | | | | UNPACK | | | | | | |
| ESD | | | | | | | AODBLD | | | |
| AOD | SELECT | | | | | | | TAGBLD | ANALYZE | ANALYZE |
| TAG | SELECT | | | | | | | | | |
| NTUP | | | | | | | | | ANALYZE | ANALYZE |

Figure 2. Identified ATLAS transformations. Categories for input and output datasets are shown the left column and upper row, respectively

ROOT and python clients are available providing the full DIAL functionality as described earlier. These and the above analysis services and the combined ntuple transformation are the basis of demos available in the latest DIAL release.

## CONCLUSIONS

ATLAS has established a framework for a distributed analysis system with continuing contributions from DIAL, GANGA and AMI. Over the next couple months ADA will define the required catalogs, provide dataset description of ATLAS production data, and add the required suite of corresponding transformations. Analysis services will be deployed at different sites to enable user-level production and analysis.

## ACKNOWLEDGEMENTS

Many useful discussions that have taken place within PPDG, ARDA and the ATLAS database and production groups. We gratefully acknowledge support from the U.S. DOE through PPDG, PPARC via GridPP, CERN, University of Bern, the IFIC computing grid group and IN2P3/CNRS.

## REFERENCES

[1] http://www.cern.ch/ATLAS.
[2] http://www.usatlas.bnl.gov/ADA.
[3] http://www.usatlas.bnl.gov/PAT.
[4] http://root.cern.ch.
[5] http://jas.freehep.org.
[6] http://www.usatlas.bnl.gov/~dladams/dial.
[7] http://ganga.web.cern.ch.
[8] http://atlasbkk1.in2p3.fr.8180/AMI.
[9] D. Adams, “AJDL: Abstract Job Description Language”, version 0.30 (2004). See ADA [2].
[10] D. Adams, “Datasets for the Grid”, version 5 (2003). See ADA [2].
[11] http://www.griphyn.org/chimera.
[12] http://www.nordugrid.org/applications/prodsys.
[13] http://lcg.web.cern.ch/LCG/peb/arda.
[14] http://www.platform.com/products/LSF.
[15] http://www.cs.wisc.edu/condor.
[16] http://egee-jra1.web.cern.ch/jra1.
[17] http://www.atlasgrid.bnl.gov/magda/info.
[18] http://mbranco.cern.ch/mbranco/cern/donquijote.
[19] http://www.pp.rhul.ac.uk/~rybkine/packagemgr.
[20] http://seal.web.cern.ch/seal